pump.io Documentation

Release 1.0.0

Jan Kusanagi, Mats Sjöberg, AJ Jordan contributors

Jul 02, 2018

User Documentation

CHAPTER 1

Basic User Guide

Pump.io is a decentralized/federated social network, composed by many servers, where users can choose where to sign up for an account, and communicate with Pump users from any other public Pump.io server.

It's currently under construction, but there are many things that can be done already. You can follow people, post notes, images and other media, and like, share or reply to other people's posts.

If you don't have a Pump account yet, click here for some help.

When you register, you'll be in front of the default web interface, but that's just the door. You can choose to interact with the Pump network using this web interface, or you can install a desktop client (or *app*), a mobile client, or interact from 3rd party web services of different kinds. Here's a comprehensive list of clients and services.

One of the first things you'll want to do is start following some people. Here's some info about that. Check out this list of users by language, where you can also add yourself.

But before that, it would be really useful to fill in some data in your profile, and uploading an avatar. Saying something about yourself in the Bio helps people find you, and makes it more likely that people with similar interests will follow you, so it makes starting interacting with people easier and faster.

It also helps to post a public "hello world" message, so people who find you have something to comment on.

If you used identi.ca in the past, take a look at this note about some of the big changes that identi.ca has undergone, now that it's part of the Pump.io network.

These are some other guides explaining how Pump.io works, how to get started, and providing usage tips by users:

- Getting Started With Pump.io by Stephen Sekula
- Who to follow, most shared, most liked... (informal stats)
- Some tips for a better experience using the Pump.io network by JanKusanagi
- FAQ - Frequently Asked Questions

CHAPTER 2

Getting started with the web interface

(TODO)

CHAPTER 3

Frequently Asked Questions

Seldomly asked questions. By real users. Not those clever developers. Here's also a basic user guide.

3.1 How do I follow a user on a remote instance?

OK - so you now have your new, shiny identi.ca account migrated to the wonderful Pump platform, or a new Pump account on any other server. In the former case, obviously all your existing Friends/Followers subscriptions from identi.ca have been seamlessly migrated, but sooner or later, you will make a new friend. Honest, you will. No, you will. And in the latter case, you'll probably want to start following someone, right?

If your new mate is also on your same server, then it's easy enough. Very similar to how it used to be on identi.ca in StatusNet times, in fact, just click 'Follow'. The 'Follow' button should magically transform to 'Following'.

Let's imagine you want to follow a user on a remote Pump instance. For example, you see an interesting note from 'david@fmrl.me'. Ensure you are logged in to your Pump account. Then visit David's Pump page at https://fmrl.me/david

Click 'Follow' as before but instead of entering your login credentials on the remote instance, click 'Account on another server ?' and type in your Webfinger identifier, such as 'harry@identi.ca'.

Click 'Authorize' when asked 'Do you want to authorize fmrl.me to access your account?' and then click 'Follow' when the user's home page is displayed.

Again, this isn't too far removed for the process for subscribing to a remote StatusNet user, but the GUI may surprise a few people at first.

As a start, you might want to check the Users by language list.

3.2 Why isn't my post available on the Internet?

If you have spent hours lovingly crafting a note on Pump and proudly posted the link to all your friends and family, there's nothing worse than people telling you 'That link you sent doesn't work'.

Pump has strict controls on privacy and publishing. By default, a note will only be posted (and visible) to your 'Followers'. If you want a note to be visible to everyone on the Internet, ensure you include 'Public' in the 'To:' or 'CC:' list.

There are enhancement requests (#364) for Pump to set defaults for your preferred distribution list or, alternatively, to remember (#431) the settings used on your last post.

Many of the other clients already have the option to always post to Public.

3.3 Can I include HTML markup in the WYSIWYG post editor?

In the web interface, no. However, some other clients support Markdown.

3.4 Is there a public timeline?

Well, yes and no. The web interface does not show a public timeline. Pump.io servers post the public notes from their users to OFireHose and it is possible to obtain a feed from there, in ActivityStreams format (JSON). So clients may present a public timeline using that feed. At this moment, Puma (client for Android) and Pumpa (desktop client) show the Firehose timeline. Web interfaces to the public timeline are in the 'Hubub' deployments (see Clients): http://hubub.e43.eu/ , http://hubub.jpope.org/ and http://ppump.redaustral.tk/rss.php (RSS feed). For more information, you can visit OFirehose.com, or track issue #656.

3.5 Are there groups like in StatusNet?

Not yet. There is an open issue about group support (#299), you can track it to know when/how it will be implemented (you can help to make it happen, too!).

Support for groups is partially implemented at the moment. Though the web interface doesn't have anything for it, it's possible to create, join and delete groups. The Dianara client has experimental support for this, but posting to a group requires manually keeping track of group ID's, and the comments to notes posted to the groups are not well distributed to all members. This still needs some work before groups are usable.

3.6 Why can't I see comments in threaded view?

By now, it's technically possible to reply to a comment, and it is displayed in the minor inbox feed (the "Meanwhile..." column), and the original poster and commenter receive a notification. However, it's not clear if it is the intended behavior, or only comments to the original post should be allowed. You can track issue #497 and participate in the discussion about this topic. https://github.com/e14n/pump.io/issues/497

3.7 What is the visibility of comments?

If I comment on someone else's post, how public is my comment?

If you comment on somebody's post, it depends on the original posts audience. If it's a public post, your comment will be public. If the post was to a limited audience, only the original author will be able to see your comment. Some of this may change a little in the future...

3.8 What is the visibility of a shared post?

If I share someone else's post, how public is my share? Is it only shared to my followers?

If you share someone else's post, again it depends on the original author's audience. If the original post was public, the reshare will be public. If the original post was to a limited audience, the reshare is supposed to only be visible to the original authors audience, but, currently, it goes out to your followers. Honestly, if the original author posts something to a limited audience, maybe it shouldn't be able to be reshared at all.

3.9 Why can't I find somebody when fulfilling the To: or CC: boxes?

You can send/CC to the people you follow.

If you are following somebody and you cannot find him/her when you type the name in the To: or CC: box, maybe you are experiencing this issue: https://github.com/e14n/pump.io/issues/805

Some of the clients will also allow you to address a post to certain people just by typing @ and selecting the user from a pop-up list. It's just another way to automatically fill the To: or CC: fields.

3.10 How can I set or change my e-mail address?

While the web interface doesn't provide a way to do this yet, the Dianara client has an option for this, since version 1.3.1.

3.11 Why can't I post a comment to a certain post?

You're probably suffering from this issue, #1027.

This basically happens when you are on server A, and post to a note from a user on server B that no one (including you) on server A is following. You might be seeing this because someone you follow shared it, but since nobody on your server follows the author of the post, the post was never really "delivered" to your server, and that results in the "no original post" error.

This error is especially unclear in the web interface. There's not really a workaround for this yet, but if you follow the author of that post, you'll be able to comment on their **future** posts.

CHAPTER 4

Clients and Services

These are some client applications and services that support Pump.io.

See also: Libraries

4.1 Web

- ih8.it - Hate button for the web.
- hip2.it - Like button.
- lurve.jpope.org - Love button.
- pump2status.net - Find your StatusNet/GNUsocial friends on the pump network.
- pump2tweet.com - Send activities from the pump network to Twitter.
- OpenFarmGame - A farming game for the pump network.
- PumpLive - Stats server.
- OFirehose - The pump network firehose.
- brdcst.it - Broadcast your blog, µblog, etc feeds to your social networks, including Pump.io.
- pump2rss.com - Generates an RSS (Atom) feed of the activity stream.
- rss.io.jpope.org - Alternate for pump2rss.com.
- pumpiostatus.website - Check the status and uptime of all registered Pump servers.
- Granada- Website, bookmarklet and button for publishing content from other places in Pump.io ("Share in Pump.io" thingy). Website in Spanish. Other deployments: http://granada.mamalibre.com.ar.
- hubub.e43.eu - View the Firehose aka what's happening in Pump.io, in real time. Source code here. Other deployments: https://hubub.polari.us.
- PumpBridge - Connects Pump.io to facebook and googleplus. Source code here.

- PPump - Ppump: RSS Feed of the Firehose and public user directory. Live instances at https://www.inventati.org/ppump and https://pump.mamalibre.com.ar.

4.2 Desktop

- **Dianara - A Qt desktop app.**
  - MSwindows builds of Dianara. Website in Spanish, for now.
- **Pumpa - Another Qt client under development.**
  - How to for building Pumpa on OS X.
- Choqok - KDE micro-blogging client.
- spigot - Console client for rate-limited (RSS) feed aggregation. Implemented in Python via PyPump.
- PumpTweet - Find notes from your Pump account, shorten them, make a URL to the original note, and post the short version as a tweet on Twitter. It can also be used for GNU Social (StatusNet).
- PumpMigrate - Move or sync contacts between Pump.io accounts.
- p - A Pump.io version of the command line utility 't'.
- NavierStokes - Allows you to bridge between social network accounts. ALPHA release.
- Pumpio-el - Pump.io client for Emacs.
- GPump - A GTK+ Pump.io client in the concept stages.
- Manivela - Command line client written in PHP. Documentation in Spanish.

4.3 Android

- Impeller - ICS (4.0) or above - Google Play / Download APK
- Puma - Download APK
- PumpFM - A simple app that scrobbles the music listened on your Android phone to a Pump.io instance (Link to binary .apk)
- AndStatus - Multiple Pump.io, GNU Social and Twitter accounts. Can work offline.

4.4 iOS

- Social Monkeys - An intuitive iOS client to manage your Pump.io social activity stream.

CHAPTER 5

Built-in CLI applications

You can use any pump.io client application you want to interact with pump.io servers. However, the `pump.io` package comes with some samples to get you started and you can find some more in the repository.

5.1 pump-register-app

First use this tool to create the credentials file

```
$ ./bin/pump-register-app  -t <APPNAME>
```

`<APPNAME>` will be the name of the client app that `pump-register-app` registers with the server.

This will create the file `~/.pump.d/<SERVER>.json` that contains your credentials.

```
{
"client_id":"XXXX",
"client_secret":"YYYYY",
"expires_at":0
}
```

It will also add an entry into the server database where you will find the clientID.

(Of course, if you use the memory Databank driver the data will be lost between server runs, and you'll need to rerun the configuration.)

5.2 pump-register-user

Use this command to register a user:

```
./bin/pump-register-user  -u <USERNAME> -p <PASSWORD>
```

5.3 pump-authorize

After you register an app, you can authorize your user to use it.

```
./bin/pump-authorize -u <USERNAME>
```

When you do that it will ask you to open a website, login and verify the value. You paste that back in and all is good.

CHAPTER 6

For system administrators

6.1 Installation instructions

npm is the recommended installation method if there isn't documentation for your specific distribution listed below. The pump.io project *strongly* recommends that users avoid source-based installs.

6.1.1 Prerequisites

You'll need four things to get started:

- node.js 6.x or higher
- npm 2.15.x or higher (4.x or above preferred)
- A database server (see below)
- The `graphicsmagick` package with the `gm` command

Note: If you're planning to use npm to install the unpublished git master version of pump.io, npm 4 or better is a requirement, not a recommendation.

Database server

pump.io uses databank package to abstract out the data storage for the system. Any databank driver should work. Couchbase, MongoDB and Redis are probably the best bets for production servers. `disk` and `memory` should only be used for testing.

If you're confused, just use the MongoDB one, `databank-mongodb`.

You can find other drivers like so:

```
$ npm search databank
```

See the installation instructions for how to install a databank driver.

Note that you also need to install and configure your database server.

Changelog

pump.io version	Change type	Change
0.3.0	New dependency	GraphicksMagick with the `gm` command is now required
5.0.0	Version change	Node.js 4.x or better is now required
6.0.0	Version change	Node.js 6.x or better is now required

6.1.2 Installation channels

This page documents where you can obtain pump.io and the different levels of support we offer for each distribution channel.

Anything not listed on this page should be considered completely unsupported by the pump.io project, and you should carefully evaluate the risk before using any such resource.

Channel	Security support	Betas available	Coordinated release schedule	Recommended for
npm registry	Yes	Yes	Direct	General use
GitHub clone	Yes	Yes	Direct	Installations which need to patch the source code
Docker image	Yes	Yes	Direct	Administrators with existing Docker infrastructure they want to use

- **Security support** means that the pump.io project guarantees it will support installations using this channel with security updates.
- **Betas available** means that you can acquire pump.io betas as well as stable releases from this channel.
- **Coordinated release schedule** indicates whether this channel will have the latest release as soon as pump.io upstream cuts a release; "direct" indicates that the pump.io project is directly responsible for releases through this channel, and "coordinated" means that pump.io will coordinate schedules with this channel to ensure that updates are timely.
- **Recommended for** describes which scenarios you should use this installation method for.

6.1.3 Generic npm-based install

Note: This is the recommended install method.

First, install the prereqisites. Package names will vary across distributions. In particular, how to install npm may vary significantly, as some distributions ship severely outdated versions of npm (or don't ship npm packages at all). Feel free to ask the community for help.

Once you have the prerequisites set up, you can install the npm package like so:

```
npm install -g pump.io
```

Depending on your configuration, you may need to prefix this command with `sudo`. At this point all the files and dependencies should be set up for you.

You can now proceed to configuring pump.io. The recommended way for an npm-based install is to use JSON configuration files. You will also almost certainly want to set up HTTPS at this point, perhaps using Certbot and Let's Encrypt.

Warning: If you're connecting your pump.io site with other software (such as federated servers or using Web clients), please note that most of them save OAuth keys based on your hostname and listening port. The following changes may make your relationships stop working.

- Change of hostname
- Change of port (from 8000 to 80 or even from HTTP to HTTPS)
- Clearing your database or clearing some tables
- Changing user nicknames

We realize that these kind of changes are normal when someone's experimenting with new software, and there are (early, tentative) plans to make the software more robust in the face of this kind of change without sacrificing security, but for now it's a good idea to decide on your "real" domain name first before making connections to other sites.

6.1.4 Generic source-based install

Warning: The pump.io project *strongly* recommends against source-based installations. They're harder to upgrade and the chances of the administrator making a mistake during installation is far higher.

Unless you have a very good reason for doing this (for example, you need to patch the source code), you should do an npm-based install instead.

Feel free to ask the community if you're not sure about your particular situation.

First, install the prereqisites. Package names will vary across distributions. In particular, how to install npm may vary significantly, as some distributions ship severely outdated versions of npm (or don't ship npm packages at all). Feel free to ask the community for help.

Once you have the prerequisites set up, you should clone the git repository:

```
git clone https://github.com/pump-io/pump.io.git
```

You can then install the dependencies using `npm`:

```
cd pump.io
npm install
npm run build
```

If you want to test the install, you can run:

```
npm test
```

These tests are run on every commit to the codebase so they won't catch actual bugs, but they may catch simple installation problems. You can run them if you want, but they take a while so this step is super optional.

You can now proceed to configuring pump.io. The recommended way for an source-based install is to use JSON configuration files. You will also almost certainly want to set up HTTPS at this point, perhaps using Certbot and Let's Encrypt.

Warning: If you're connecting your pump.io site with other software (such as federated servers or using Web clients), please note that most of them save OAuth keys based on your hostname and listening port. The following changes may make your relationships stop working.

- Change of hostname
- Change of port (from 8000 to 80 or even from HTTP to HTTPS)
- Clearing your database or clearing some tables
- Changing user nicknames

We realize that these kind of changes are normal when someone's experimenting with new software, and there are (early, tentative) plans to make the software more robust in the face of this kind of change without sacrificing security, but for now it's a good idea to decide on your "real" domain name first before making connections to other sites.

6.1.5 Setup on Debian "jessie" stable

We're going to install pump.io from the npm registry, use MongoDB for the database and proxy pump.io behind the nginx web server.

Install basic requirements

Install the requirements via the Debian package manager.

```
$ sudo apt-get install mongodb nginx graphicsmagick git
```

Debian jessie provides nodejs of version 0.10, while pump.io after version 2.0.0 beta 1 requires nodejs 4.x. Therefore, it must be updated from upstream.

```
$ curl -sL https://deb.nodesource.com/setup_4.x | sudo bash -
$ sudo apt-get install -y nodejs
```

You may want to also consider running `sudo rm -r /usr/local/lib/node_modules` to remove all currently-installed npm packages. Blowing away everything already installed is optional but strongly recommended as npm v1 (which is shipped by Debian) is prone to creating broken, half-installed directory structures that interfere with future installations. In addition, Debian's npm has a default install path of `/usr/local/lib/node_modules`, whereas upstream installs to `/usr/lib/node_modules`. Simply removing the former path makes everything much less confusing.

If you want to keep your currently-installed packages the best way to do so is to save the list of installed packages, proceed with the removal, then reinstall using upstream npm. Conveniently, you can get a list of installed packages by doing `npm ls -g --depth=0 --prefix=/usr/local`.

Install and set up pump.io

Next, we create a `pumpio` user and group and create the needed folders:

```
$ sudo groupadd pumpio
$ sudo useradd -d /var/lib/pumpio -m -r -g pumpio pumpio

$ sudo mkdir -p /srv/pumpio/uploads
$ sudo chown -R pumpio:pumpio /srv/pumpio

$ sudo mkdir /var/log/pumpio/
$ sudo chown pumpio:pumpio /var/log/pumpio/
```

Next, install pump.io itself, along with the MongoDB Databank driver:

```
$ sudo npm install -g pump.io databank-mongodb@0.19.2
```

Finally pump.io needs a configuration file in `/var/lib/pumpio/.pump.io.json` with the following content (adapted to your situation):

```
{
    "driver":  "mongodb",
    "params": {"host":"localhost","dbname":"pumpio"},
    "hostname":  "your.pump.com",
    "address": "127.0.0.1",
    "port": 8080,
    "urlPort": 443,
    "secret":  "somerandomstringhere",
    "key":  "/path/to/your/ssl-cert.crt",
    "cert":  "/path/to/your/ssl-cert.key",
    "noweb":  false,
    "site":  "your.pump.com",
    "owner":  "Your Name",
    "ownerURL":  "http://your.site.com/",
    "nologger": false,
    "logfile": "/var/log/pumpio/pumpio.log",
    "serverUser":  "pumpio",
    "uploaddir": "/srv/pumpio/uploads",
    "debugClient": false,
    "firehose": "ofirehose.com",
    "disableRegistration": true,
    "noCDN": true,
    "requireEmail": false,
    "compress": true,
    "smtpserver": "localhost",
    "proxyWhitelist": ["avatar3.status.net", "avatar.identi.ca", "secure.gravatar.com
↪"]
}
```

In particular you need to replace `your.pump.com` which your actual domain name.

Warning: If you're connecting your pump.io site with other software (such as federated servers or using Web clients), please note that most of them save OAuth keys based on your hostname and listening port. The following changes may make your relationships stop working.

- Change of hostname
- Change of port (from 8000 to 80 or even from HTTP to HTTPS)
- Clearing your database or clearing some tables
- Changing user nicknames

> We realize that these kind of changes are normal when someone's experimenting with new software, and there are (early, tentative) plans to make the software more robust in the face of this kind of change without sacrificing security, but for now it's a good idea to decide on your "real" domain name first before making connections to other sites.

Make sure that the pumpio user can access your SSL certs.

You can now try and see if it works:

```
$ sudo -u pumpio pump
```

This starts up pump.io running on port 8080, this will be proxied behind nginx which will serve it on port 443 (https).

Setup nginx proxying

Create the file `/etc/nginx/sites-available/pump` with the following content (adapted to your situation):

```
upstream pumpiobackend {
  server 127.0.0.1:8080 max_fails=3;
}

server {
  server_name your.pump.com;
  rewrite ^ https://your.pump.com$request_uri?;
}

server {
  listen 443 ssl;
  server_name your.pump.com;

  ssl_certificate  /path/to/your/ssl-cert.crt;
  ssl_certificate_key  /path/to/your/ssl-cert.key;

  access_log /var/log/nginx/pumpio.access.log;
  error_log /var/log/nginx/pumpio.error.log;

  client_max_body_size 500m;

  keepalive_timeout 75 75;
  gzip_vary off;

  location / {
    proxy_http_version 1.1;
    proxy_set_header Host $http_host;
    proxy_set_header Upgrade $http_upgrade;
    proxy_set_header Connection "upgrade";
    proxy_set_header X-Real-IP $remote_addr;

    proxy_redirect off;

    proxy_buffers 16 32k;
    proxy_cache off;
    proxy_connect_timeout 60s;
    proxy_read_timeout 60s;
    proxy_pass https://pumpiobackend;
```

(continues on next page)

(continued from previous page)

```
    proxy_pass_header Server;
  }
}
```

You will of course have to provide your own SSL cert and put them in the right path. Make sure that the certificate key is not world readable! Currently the easiest (and cheapest) way to get such a certificate is by using Certbot and Let's Encrypt.

Finally you can link the file you created to enable it (in a root shell):

```
# cd /etc/nginx/sites-enabled
# ln -s ../sites-available/pump .
```

If everything is set up correctly you should now be able to restart nginx and access the site in your web browser:

```
$ sudo systemctl restart nginx
```

Before you start using your site make sure have settled on the correct hostname. You can't switch that around later without breaking federation badly.

Start pump.io automatically using systemd

You probably want to make sure pump.io is started at boot and monitored thereafter. You can accomplish this with systemd, and pump.io ships a systemd file specifically for this purpose.

To use said systemd file, see using the upstream pump.io systemd unit.

If everything seems to be working, you're done! Congratulations!

Upgrading pump.io

If you later wish to upgrade your pump.io server software, e.g. to the latest version on npm (or even to git master), you can do something like the following. (Note: you might want to backup your database first using `mongodump`, see the migration notes below.)

First, stop pump.io and switch to the pumpio user.

```
$ sudo systemctl stop pump.io@mongodb
```

Next, install the latest version with npm:

```
$ sudo npm install -g pump.io
```

It's a good idea to update the database stuff at the same time:

```
$ sudo npm install -g databank-mongodb@0.19.2
```

If you're unsure, especially after a big upgrade, you can always test it first directly by running `sudo -u pumpio pump` as above. If all seems well you can Ctrl-C and start it for real.

```
$ sudo systemctl start pump.io@mongodb
```

6.1.6 About Docker images

The pump.io project provides official support for Docker through images published on Docker Hub. Docker images are built automatically by Travis CI and will be available within 24 hours of an npm release.

Consider using the Docker images if you have existing Docker infrastructure or experience. For installs where the administrator doesn't already have a preference for Docker, we recommend npm-based installation.

This document assumes the reader is already familiar with Docker.

Installation

The Docker images are published to `pumpio/pump.io` on Docker Hub. You can pull the latest stable, for example, with:

```
$ docker pull pumpio/pump.io:stable
```

All release channels are available on Docker Hub. Each image is tagged with its exact version identifier, and the latest stable release, beta release, and nightly `master` builds are tagged with `stable`, `beta`, and `alpha` respectively. Each individual release is also tagged appropriately; the Docker tag names are the exact same as the version strings found in npm releases.

Note that if you choose to run alphas, the same considerations apply as with npm-based installs.

Configuration

The Docker images are designed to be configured via environment variables. Note that there are a couple differences in Docker images' configuration, some of which change the defaults listed in the configuration value reference.

Differences from npm builds

pump.io Docker images differ from npm-based installations in the following ways:

- The daemon listens to port 80 by default
- `datadir` is set to `/var/local/pump.io` by default
- Logs are sent to stdout by default, not stderr
- Databank drivers for MongoDB, LevelDB, Redis, memcached, and `databank-disk` are already included
- The image is configured to run `pump` as an unprivileged `pumpio` user in the container

Note that `/var/local/pump.io` is configured to be a Docker volume, which is why `datadir` is set there by default. The default for `enableUploads` (`false`) has not changed, however, so the admin is free to leave uploads disabled.

> **Warning:** In the future we may introduce important features that are automatically enabled when the `datadir` is set. These features will not be marked semver-major, so if you want to run the Docker image without the `datadir` volume mounted, you should explicitly set `datadir` to the empty string in the environment.

Security support

Docker images have the same security lifecycle as regular pump.io releases; you can find this policy on the project wiki.

When using a regular npm-based install you need to make sure that you're running a secure version of pump.io. When using Docker images, you *also* need to make sure the image's native dependencies are fully patched. To help with this, the pump.io project automatically publishes new images within 24 hours of package upgrades becoming available. For this reason, you are encouraged to regularly pull newer pump.io images from Docker Hub. You can use the version number tags to automate this so that you don't unexpectedly get new pump.io versions (which may have breaking changes).

Warning: When security issues in pump.io itself are fixed, the version number increments which results in Docker images being published under a different tag - meaning that once a patch release is available, the 24-hour automated builds switch to that patch release.

Consequently, it's still important to keep track of pump.io security releases even if you have automation keeping your Docker images up-to-date.

As an example, say you're running pump.io 5.2.0 Docker images, and you have cron configured to automatically pull newer 5.2.0 images from Docker Hub. A (fictional) security issue arises in a native dependency. In that case, you'll be fine - a newer image will be published within 24 hours, and your cronjob will automatically pick that up.

But let's say the security issue required a fix in the pump.io codebase, not a native dependency. In that situation, the project would release 5.2.1 as a security release and you would need to upgrade your Docker image. If you didn't, not only would you not have the pump.io fix, any future native dependency vulnerability fixes wouldn't be pulled in by your cronjob. Upgrading to 5.2.1 images would resolve both of these problems.

Note: We many not turn off the infrastructure maintaining old release series immediately. If our security policy says a release is unsupported, you should consider it unmaintained even if new Docker images are still being published to the relevant tag.

If you have questions about this information, or if something is confusing, please don't hesitate to contact the community.

6.2 Configuration details

6.2.1 Configuration value reference

Warning: If you're connecting your pump.io site with other software (such as federated servers or using Web clients), please note that most of them save OAuth keys based on your hostname and listening port. The following changes may make your relationships stop working.

- Change of hostname
- Change of port (from 8000 to 80 or even from HTTP to HTTPS)
- Clearing your database or clearing some tables
- Changing user nicknames

> We realize that these kind of changes are normal when someone's experimenting with new software, and there are (early, tentative) plans to make the software more robust in the face of this kind of change without sacrificing security, but for now it's a good idea to decide on your "real" domain name first before making connections to other sites.

The default config values are stored in the source file `lib/defaults.js`.

Here are the main configuration keys.

Name	Type	Description
driver	String	The databank driver you're using. Defaults to "memory", which is probably going to be terr
params	Object	Databank driver params; see the databank driver README for details on what to put here.
hostname	String	The hostname of the server. Defaults to "127.0.0.1" which doesn't do much for you.
address	String	The address to listen on. Defaults to `hostname`, which is OK for most systems. Use this if
port	Number	Port to listen on. Defaults to 31337, which is no good. You should listen on 80 or 443 if you
urlPort	Number	Port to use for generating URLs. Defaults to the same as `port`, but if you're insisting on pr
bounce	Boolean	If `true`, set up a mini-server on port 80 that redirects to HTTPS
secret	String	A session-generating secret, server-wide password.
noweb	Boolean	Hide the Web interface. Set this to something truthy to disable the Web interface.
site	String	Name of the server, like "My great social service".
owner	String	Name of owning entity, if you want to link to it.
ownerURL	String	URL of owning entity, if you want to link to it.
appendFooter	String	A bit of custom HTML you want appended to the footer text.
mainImage	String	An image/icon for the main page; could be an external URL or an image in the "public/" fol
nologger	Boolean	Turn off logging, if you're debugging or whatever.
logfile	String	Full path to the logfile. Logs are JSON in bunyan format.
logLevel	String	Log level used by bunyan (see bunyan loglevels documentation)
serverUser	String	If you're listening on a port lower than 1024, you need to be root. Set this to the name of a u
key	String	If you're using SSL, the path to the server key, like "/etc/ssl/private/myserver.key".
cert	String	If you're using SSL, the path to the server cert, like "/etc/ssl/private/myserver.crt".
hsts	Boolean | Object	Controls the HTTP `Strict-Transport-Security` header. It's passed directly to the
datadir	String	Directory for the server to store data in (mostly uploads). Should be the full path of a local d
enableUploads	Boolean	If you want to enable file uploads, set this to `true`. Make sure that `datadir` is set and tha
controlSocket	String	Path to a Unix domain socket that can be used to control and query the master process.
debugClient	Boolean	For developers, if you're debugging the Web interface and you want to use the non-minified
firehose	String	Firehose host running the ofirehose software. Public notices will ping this firehose server an
spamhost	String	Host running activityspam software to use to test updates for spam.
spamclientid	String	OAuth pair for spam server.
spamclientsecret	String	OAuth pair for spam server.
disableRegistration	Boolean	Disables registering new users on the site through the Web or the API.
noCDN	Boolean	Use local copies of the JavaScript libraries instead of the ones on the CDN. Good for debug
requireEmail	Boolean	Require an email address to register. Should be ignored if email server isn't configured.
smtpserver	String	Server to use for sending transactional email. If it's not set up, no email is sent and features
smtpport	Number	Port to connect to on SMTP server. Defaults to 25 which is really the only sane value.
smtpuser	String	Username to use to connect to SMTP server. Might not be necessary for some servers.
smtppass	String	Password to use to connect to SMTP server. Might not be necessary for some servers.
smtpusetls	Boolean	Try to negotiate using SSL with the SMTP server. Defaults to `true`, because it's a smart id
smtpusessl	Boolean	Only use SSL with the SMTP server. You may need to change the `smtpport` value if you
smtptimeout	Number	Timeout for connecting to the SMTP server in milliseconds. Change this if... I dunno. I se
smtpfrom	String	Email address to use in the "From:" header of outgoing notifications
compress	Boolean	Use gzip or deflate to compress text output. This can cut down on network transfers conside

Name	Type	Description
children	Number	Number of children worker processes to run
clients	Array of Objects	You can pre-configure some OAuth credentials if you want to have a replicable configuratio
sockjs	Boolean	Use SockJS-node to provide a realtime connection
cleanupSession	Number	Time interval to clean up sessions (in ms). These are staggered a bit if you have more than c
cleanupNonce	Number	Time interval to clean up OAuth nonces (in ms). Staggered.
favicon	String	Local filesystem path to the favicon.ico file to use. This will be served as "/favicon.ico" by t

And here are all obsolete configuration keys.

Name	Type	Description	Removed in	Removal issue
uploaddir	String	If you want to enable file uploads, set this to the full path of a local directory. It should be writeable and readable by the `serverUser`.	3.0	#1261

These values can be set via CLI flags, environment variables, or a JSON configuration file. See the individual documentation for each of these methods for details.

6.2.2 Configuration via CLI flags

The `pump` daemon accepts configuration values via options passed directly on the commandline. Flags start with `--` - for example, the `port` key can be set by passing `--port`. Camelcasing like `urlPort` should be replaced with `-` (i.e. `urlPort` becomes `--url-port`). Keys whose value is an object can be specified using `.` to separate nested keys. For example, the `params` key set to the following object:

```
{ "host": "localhost" }
```

can be set by passing `--params.host localhost`.

CLI flags always take precedence over everything else, including environment variables and JSON configuration files.

For a list of all available configuration values, see the reference list.

6.2.3 Configuration via environment variables

Note: This is the recommended configuration method for containerized pump.io daemons.

The `pump` daemon accepts configuration values via environment variables. Each variable is prefixed with `PUMPIO_` and then has the capitalized configuration key you want to set. For example, the `port` key can be set via the environment variable `PUMPIO_PORT`.

To configure camelcased config values like `urlPort`, replace the camelcasing with an underscore (_). For example, `urlPort` would become `PUMPIO_URL_PORT`. Keys whose value is an object can be specified using __ (two underscores) to separate subkeys. For example, the `params` key with the following object as its contents:

```
{ "host": "localhost" }
```

can be represented by exporting `PUMPIO_PARAMS__HOST` to the environment with a value of `localhost`. Exactly how to set environment variables will depend on how you start the pump.io daemon.

Environment variables override JSON configuration files and are overridden by CLI flags.

For a list of all available configuration values, see the reference list.

6.2.4 Configuration via JSON configuration files

Note: This is the recommended configuration method for non-containerized (standalone) pump.io daemons.

pump.io can use a JSON file for configuration. `pump` will look for configuration files at `/etc/pump.io.json` and `~/.pump.io.json`. These files are expected to have an object at the top level which has key-value pairs for each configuration option. The `pump.io.json.sample` file should give you an idea of how to use it.

You can also override the config file location with the `-c` option:

```
pump -c <CONFIG_FILE>
```

Configuration files are overridden by CLI flags and environment variables.

For a list of all available configuration values, see the reference list.

6.2.5 Running the daemon

The recommended way to run the pump.io daemon is to use the systemd service file that's shipped with the package. This service file automatically sets many important options, such as the `NODE_ENV` environment variable (which impacts performance) and several security-related systemd settings. See "Using the upstream pump.io systemd unit" for detailed instructions, and note that you can override parts of it if you need using drop-in files.

If you can't use the systemd unit, you need to arrange to run either `./bin/pump` in your git clone (if you installed from souce) or `pump` (if you installed from npm). You'll probably get a more reliable experience if you use your operating system's built-in `init` service manager, or you could use forever on npm to keep the daemon running.

6.2.6 Set `NODE_ENV` for better performance

`NODE_ENV` determines the environment pump.io is running in. This should be set to `production` in production environments or performance will be *significantly* degraded.

In development environments it should be set to `development`, which is the default.

How to set this environment variable will depend on how you start the pump.io daemon. If you use the systemd unit shipped with the package, `NODE_ENV` will be set to `production` for you automatically.

6.2.7 Using Certbot and Let's Encrypt for HTTPS in pump.io

After you have set up pump.io - but before you staring interacting with other servers or clients - it's a good idea to set up TLS (HTTPS), which prevents others from reading or tampering with users' private data. It also helps keep the overall pump.io network secure since other servers can deliver activities over TLS, giving their users the same confidentiality and integrity guarantees. Changing ports or from HTTP to HTTPS after interacting with other servers or clients can cause problems. The npm-based install instructions talk about other changes that may also cause problems.

Let's Encrypt is an automated Certificate Authority which will issue TLS certifiates for free. This document will show you how to set up Certbot, a popular Let's Encrypt client, to acquire and automatically renew Let's Encrypt certificates.

Note: These instructions do not work with Docker images.

First, install Certbot. Your GNU/Linux distribution may already have it packaged for easy installation, or you may need to do that yourself - either way, the Certbot website will tell you how to get it installed.

Once Certbot is installed run this command, inserting your actual email address and domain names:

```
$ certbot certonly --email user@host.root --webroot -w /usr/lib/node_modules/pump.io/
↪public/ -d domain.tld,www.domain.tld
```

The email address provided will receive notifications when the certificate is due to expire which can be helpful as a reminder.

Note that `/usr/lib/node_modules/pump.io/` is usually where npm installs pump.io when you use an npm-based installation. If your setup is different, provide the full path to your pump.io installation, plus `/public/` at the end.

Next, edit your pump.io configuration (for example, `/etc/pump.io.json` if you're using a JSON file for configuration) and be sure that it contains these settings (again replacing `domain.tld` with your production domain):

```
"hostname":  "domain.tld",
"key": "/etc/letsencrypt/live/domain.tld/privkey.pem",
"cert": "/etc/letsencrypt/live/domain.tld/fullchain.pem",
"port":  443,
"bounce": true,
"hsts": true,
```

Restart the pump.io software. Congratulations, you should now have TLS working!

To automate renewals you can make a cron job or systemd timer that runs `certbot renew`. That should take care of renewing the certificate but the email address provided during the initial setup can be helpful as a fallback method in case something happens. The pump.io software should also be restarted once the certificate is renewed. This can be accomplished with zero-downtime restarts and Certbot pre/post validation hooks.

For more on `certbot renew`, consult the Certbot documentation.

6.2.8 Self-signed certs and CACert non-support in pump.io

Over the years there have been some requests to support self-signed certificates or CACert certificates in pump.io. Self-signed certificates and certificates signed by less-trusted providers like CACert appear to work when you're installing, but later your server will have trouble communicating with other pump.io servers.

This trouble occurs because while *you* may have CACert or your self-signed certificatein your operating system's trust store, other pump.io servers will not. When your server tries to deliver posts to them, or they to you, it won't work because the other servers won't trust your server and distribution will fail.

Support for these certs will never appear in pump.io core. Not only would it be a burden to maintain, it would compromise the security of the overall network - and for no good reason since reasonable alternatives exist.

If you are looking for a free TLS certificate, please consider Let's Encrypt. Documentation is available on how to use Certbot to automatically get Let's Encrypt certificates for pump.io.

6.2.9 Using a web server proxy

pump.io is designed to be a standalone server. You do not need to set up an Apache or nginx or lighttpd Web server in front of it. In fact, that's going to make things harder for you:

- Stuff like WebSockets is going to work less well
- It will makes things harder to debug
- Future planned pump.io features, like automatic, out-of-the-box HTTPS management may not work as intended

The pump.io project strongly recommends against doing this if at all possible.

If you really insist, check the configuration options carefully. If you want http://pump.yourdomain.example/ to proxy to the pump.io daemon listening on port 8000 on 127.0.0.1, use configuration options like this:

```
"hostname": "pump.yourdomain.example",
"urlPort": 80,
"address": "127.0.0.1",
"port": 8000
```

6.3 Upgrade instructions

6.3.1 Upgrading to pump.io 2.x from 1.x

These are the instructions for upgrading pump.io 1.x to 2.x. They will work for any release in the 1.x series and can be used to upgrade to any release in the 2.x series, including beta releases.

Note: If you're on pump.io 0.3.0 or below, it's recommended to upgrade to pump.io 1.x before attempting to upgrade to 2.x. It's not strictly necessary, but it makes things easier if things go wrong since there's less changes that could be causing problems.

If at any point you run into trouble, contact the community and they'll be happy to sort you out.

For npm-based installs

First, check if you have any plugins configured. If you haven't defined a `plugins` key in your `pump.io.json`, then you don't have any plugins installed. If you *do* have plugins installed, see "Migrating plugins" below.

To run the upgrade, invoke:

```
$ sudo npm install -g pump.io@2
```

Note: If you're trying to upgrade to the latest beta, specify `pump.io@beta` instead.

Complete your upgrade by restarting your pump.io process.

For source-based installs

First, check if you have any plugins configured. If you haven't defined a `plugins` key in your `pump.io.json`, then you don't have any plugins installed. If you *do* have plugins installed, see "Migrating plugins" below.

Next, fetch new changes, prefixing `sudo` as necessary:

```
$ cd /path/to/your/installation/directory
$ git fetch
```

If you've modified templates, you need to save the changes you've made. You can check to see whether you've made changes by running `git status` and `git log` - if you have, save them to a file.

Now, discard the changes you've just saved:

```
$ git checkout .
```

At this point, `git status` should report a working directory with no modified files.

Now that we're prepared, we'll perform the upgrade itself by checking out the relevant tag. You'll have to check what the latest version in the 2.x series is using `git tag` - in this case, we're assuming that 2.0.5 is the latest available.

```
$ git checkout v2.0.5
$ npm install
```

At this point, if you had previously saved template modifications, you should now use the information you saved to separate files to restore your modifications. Because Jade's syntax differs significantly from utml's you won't be able to directly apply the saved changes; however, you should have saved enough information to reintroduce the same semantic changes.

If you just restored template changes, it's recommended that you run the linting process to ensure that your changes are high-quality and consistent with surrounding code:

```
$ npm run lint:jade
```

Finally, regardless of whether or not you modified templates, you need to rebuild client-side template resources which aren't checked into git:

```
$ npm run build
```

Complete your upgrade by restarting your pump.io process.

Migrating plugins

pump.io 2.x upgrades the Express web framework to the 3.x series, which contains breaking changes. This affects you because plugins essentially get access to internal application details, including the Express application object. This means that any plugin that relies on the behavior of the app object is likely to be affected by Express 3.x's breaking changes. See the Express 3.x change log for details.

Fixing this can be done in any number of ways - which way you choose will depend on the nature and complexity of the plugin. For simpler plugins, it may make sense to simply examine the plugin and compare all uses of `app` with the Express change log. Alternately, it might be easier to simply start the app, try out behaviors that your plugin affects or creates, and see what crashes.

6.3.2 Upgrading to pump.io 3.x from 2.x

These are the instructions for upgrading pump.io 2.x to 3.x. They will work for any release in the 2.x series and can be used to upgrade to any release in the 3.x series, including beta releases.

Note: If you're on pump.io 1.x or below, it's recommended to upgrade to pump.io 2.x before attempting to upgrade to 3.x. It's not strictly necessary, but it makes things easier if things go wrong since there's less changes that could be causing problems.

If at any point you run into trouble, contact the community and they'll be happy to sort you out.

For npm-based installs

pump.io 3.x obsoletes the `uploaddir` configuration option and replaces it with `datadir` and `enableUploads`. For details, see issue #1261.

If your `pump.io.json` does not include an `uploaddir` configuration value, you're unaffected by this change. Skip to the `npm install` command below.

If it does, you'll need to migrate to the `datadir` configuration option. See "Migrating to `datadir`" below.

To run the upgrade, invoke:

```
$ sudo npm install -g pump.io@3
```

Note: If you're trying to upgrade to the latest beta, specify `pump.io@beta` instead.

Complete your upgrade by starting or restarting your pump.io process.

For source-based installs

pump.io 3.x obsoletes the `uploaddir` configuration option and replaces it with `datadir` and `enableUploads`. For details, see issue #1261.

If your `pump.io.json` does not include an `uploaddir` configuration value, you're unaffected by this change. Skip to the `npm install` command below.

If it does, you'll need to migrate to the `datadir` configuration option. See "Migrating to `datadir`" below.

There are several steps needed to run an upgrade. First, fetch new changes:

```
$ cd /path/to/your/installation/directory
$ sudo git fetch
```

If you've modified templates, you need to save the changes you've made. You can check to see whether you've made changes by running `git status` and `git log` - if you have, save them to a file.

Note: If you're a proficient git user, you may find it easier to ignore these instructions and instead commit your changes, checkout the new tag, then cherry-pick the commit you just made.

Now, discard the changes you've just saved:

```
$ sudo git checkout .
```

At this point, `git status` should report a working directory with no modified files.

Now that we're prepared, we'll perform the upgrade itself by checking out the relevant tag. You'll have to check what the latest version in the 2.x series is using `git tag` - in this case, we're assuming that 2.0.5 is the latest available.

```
$ sudo git checkout v2.0.5
$ sudo npm install
```

At this point, if you had previously saved template modifications, you should now use the information you saved to separate files to restore your modifications. The templates haven't changed significantly, so a line-by-line diff will probably apply directly with little to no conflicts.

If you just restored template changes, it's recommended that you run the linting process to ensure that your changes are high-quality and consistent with surrounding code:

```
$ npm run lint:jade
```

Finally, regardless of whether or not you modified templates, you need to rebuild client-side template resources which aren't checked into git:

```
$ sudo npm run build
```

Complete your upgrade by starting or restarting your pump.io process.

Migrating to `datadir`

Otherwise, you need to migrate your storage directory. First, shut down your pump.io server. How to do this will vary depending on what supervisor you use to manage it. For example, if you're using systemd with a unit named `pump.io.service`, you might do:

```
$ sudo systemctl stop pump.io
```

Next, determine if your `uploaddir` ends in `/uploads`. If so, all you have to do is edit `uploaddir` to be named `datadir`, remove `/uploads` from the end of the path, and add a new line that sets `enableUploads` to `true`. After these steps, you're done migrating and can skip to the `npm install` command below.

If your `uploaddir` does *not* end in `/uploads`, your migration process is a little more complicated. First, move your old uploads directory somewhere else. For example, if it was stored at `/var/lib/pumpio`, you might do:

```
$ sudo mv /var/lib/pumpio /var/lib/pumpio.bak
```

Next, create a new directory where your uploads directory used to be, setting the owner and group to your `serverUser` (assumed to be `pumpio` in this example):

```
$ sudo mkdir /var/lib/pumpio
$ sudo chown pumpio:pumpio /var/lib/pumpio
```

Next, move your old uploads directory back into the new directory, giving it the name `uploads`. For example:

```
$ sudo mv /var/lib/pumpio.bak /var/lib/pumpio/uploads
```

The last step in the migration is editing your `pump.io.json`. Find the `uploaddir` option and change it to use `datadir`. Then, add a new configuration value called `enableUploads` and set it to `true`.

You're now done migrating to `datadir` and should return to where you were before in this document.

6.3.3 Upgrading to pump.io 4.x from 3.x

These are the instructions for upgrading pump.io 3.x to 4.x. They will work for any release in the 3.x series and can be used to upgrade to any release in the 4.x series, including beta releases.

Note: If you're on pump.io 2.x or below, it's recommended to upgrade to pump.io 3.x before attempting to upgrade to 4.x. It's not strictly necessary, but it makes things easier if things go wrong since there's less changes that could be causing problems.

If at any point you run into trouble, contact the community and they'll be happy to sort you out.

For npm-based installs

First, check if you have any plugins configured. If you haven't defined a `plugins` key in your `pump.io.json`, then you don't have any plugins installed. If you *do* have plugins installed, see "Migrating plugins" below.

To run the upgrade, invoke:

```
$ sudo npm install -g pump.io@4
```

Note: If you're trying to upgrade to the latest beta, specify `pump.io@beta` instead.

Complete your upgrade by restarting your pump.io process.

Note that the logic that loads the configuration was rewritten to be more powerful. While it's expected that pump.io will behave the same, there could be edge cases impacting your setup. Therefore, please double-check that all your configuration values are still being respected.

For source-based installs

First, check if you have any plugins configured. If you haven't defined a `plugins` key in your `pump.io.json`, then you don't have any plugins installed. If you *do* have plugins installed, see "Migrating plugins" below.

Next, fetch new changes, prefixing `sudo` as necessary:

```
$ cd /path/to/your/installation/directory
$ git fetch
```

If you've modified templates, you need to save the changes you've made. You can check to see whether you've made changes by running `git status` and `git log` - if you have untracked changes reported by `git status`, save them with `git stash`; if you have committed changes reported by `git log`, make a note of the commit ids.

Now that we're prepared, we'll perform the upgrade itself by checking out the relevant tag. You'll have to check what the latest version in the 4.x series is using `git tag` - in this case, we're assuming that 4.0.0 is the latest available.

```
$ git checkout v4.0.0
$ npm install
```

At this point, if you previously ran `git stash`, you should run `git stash apply`. If you made a note of commits, you should tell Git to apply those commits on the new checkout using `git cherry-pick`.

If you just restored template changes, it's recommended that you run the linting process to ensure that your changes are high-quality and consistent with surrounding code:

```
$ npm run lint:jade
```

Finally, regardless of whether or not you modified templates, you need to rebuild client-side template resources which aren't checked into git:

```
$ npm run build
```

Complete your upgrade by restarting your pump.io process.

Note that the logic that loads the configuration was rewritten to be more powerful. While it's expected that pump.io will behave the same, there could be edge cases impacting your setup. Therefore, please double-check that all your configuration values are still being respected.

Migrating plugins

pump.io 4.x upgrades the Express web framework to the 4.x series, which contains breaking changes. This affects you because plugins essentially get access to internal application details, including the Express application object. This means that any plugin that relies on the behavior of the app object is likely to be affected by Express 4.x's breaking changes. See the Express 4.x change log for details.

Fixing this can be done in any number of ways - which way you choose will depend on the nature and complexity of the plugin. For simpler plugins, it may make sense to simply examine the plugin and compare all uses of `app` with the Express change log. Alternately, it might be easier to simply start the app, try out behaviors that your plugin affects or creates, and see what crashes.

6.3.4 Upgrading to pump.io 5.x from 4.x

These are the instructions for upgrading pump.io 4.x to 5.x. They will work for any release in the 4.x series and can be used to upgrade to any release in the 5.x series, including beta releases.

Note: If you're on pump.io 3.x or below, it's recommended to upgrade to pump.io 4.x before attempting to upgrade to 5.x. It's not strictly necessary, but it makes things easier if things go wrong since there's less changes that could be causing problems.

If at any point you run into trouble, contact the community and they'll be happy to sort you out.

For npm-based installs

First, check what Node.js version you're using. Unless you've set up a Node version manager, `node --version` should tell you this information. If you're using Node 4 or above, you'll be able to continue running pump.io without issues. If you're using Node 0.10 or 0.12, see "Upgrading Node.js" below.

Next, check if your `pump.io.json` contains an `appendFooter` option. If it does, see "Adjusting appendFooter" below.

Finally, the systemd unit file has enabled many security options. If you have `datadir` set and its value points to somewhere in `/home` or `/root`, these security restrictions will impact you. See "Dealing with systemd changes" below.

It is expected that these changes won't impact any other configurations, but this change is complicated and administrators should allow for extra time to deal with unexpected issues.

After doing this, you should be ready to upgrade your installation.

To run the upgrade, invoke:

```
$ sudo npm install -g pump.io@5
```

Note: If you're trying to upgrade to the latest beta, specify `pump.io@beta` instead.

Complete your upgrade by restarting your pump.io process.

For source-based installs

First, check what Node.js version you're using. Unless you've set up a Node version manager, `node --version` should tell you this information.

If you're using Node 4 or above, you'll be able to continue running pump.io without issues. If you're using Node 0.10 or 0.12, see "Upgrading Node.js" below.

Next, check if your `pump.io.json` contains an `appendFooter` option. If it does, see "Adjusting appendFooter" below.

Then, check if you're impacted by changes to the systemd unit file, which has enabled many security options. If you have `datadir` set and its value points to somewhere in `/home`, or if your installation's source code is stored in `/home`, these security restrictions will impact you. See "Dealing with systemd changes" below.

It is expected that these changes won't impact any other configurations, but this change is complicated and administrators should allow for extra time to deal with unexpected issues.

After doing this, you should be ready to upgrade your installation.

To start the upgrade, fetch new changes, prefixing `sudo` as necessary:

```
$ cd /path/to/your/installation/directory
$ git fetch
```

If you've modified templates, you need to save the changes you've made. You can check to see whether you've made changes by running `git status` and `git log` - if you have untracked changes reported by `git status`, save them with `git stash`; if you have committed changes reported by `git log`, make a note of the commit ids.

Now that we're prepared, we'll perform the upgrade itself by checking out the relevant tag. You'll have to check what the latest version in the 5.x series is using `git tag` - in this case, we're assuming that 5.0.0 is the latest available.

```
$ git checkout v5.0.0
$ npm install
```

At this point, if you previously ran `git stash`, you should run `git stash apply`. If you made a note of commits, you should tell Git to apply those commits on the new checkout using `git cherry-pick`.

If you just restored template changes, you'll need to check that they still work. The icon set has been switched from Bootstrap's Glyphicons to Font Awesome, so you should adjust any icons used in your modifications to match. Underscore (the _ variable) has been replaced with Lodash, a faster implementation that is mostly (but not entirely) compatible; if your templates use _ in inline JavaScript, you'll need to check that they still work with Lodash.

Additionally, 5.x upgrades Backbone to the 1.3.3 release. If you modified web UI JavaScript, please ensure that it still works with the newer version of Backbone.

It's recommended that you run the linting process to ensure that your changes are high-quality and consistent with surrounding code:

```
$ npm run lint:jade
```

Finally, regardless of whether or not you modified templates, you need to rebuild client-side template resources which aren't checked into git:

```
$ npm run build
```

Complete your upgrade by restarting your pump.io process.

Upgrading Node.js

As previously announced, pump.io 5.0 drops support for Node 0.10 and 0.12. These versions already operated with degraded cross-site scripting security protections and were preventing important maintenance work.

Most mainstream distributions ship supported versions of Node.js, so if possible, you should upgrade your distribution. If this is not an option, you can use NodeSource to get a newer version of Node.

If you aren't sure how to move forward, contact the community and they'll help you sort through your options.

Adjusting `appendFooter`

pump.io adds a period and a space after the text in the footer, before `appendFooter`'s value is interpolated.

To account for this, simply remove these characters from the beginning of `appendFooter` in your `pump.io.json`, or adjust `appendFooter` some other way if you didn't previously have these characters.

Dealing with systemd changes

As of pump.io 5.x, the systemd unit shipped with the package uses the `InaccessiblePaths=` option to prevent the daemon from accessing several common directories in `/`. It also restricts access to miscellaneous other system resources and, perhaps most importantly, sets `ProtectHome=true`. For more information, refer to systemd.exec(5).

This is done to reduce attack surface and limit potential damage in the event that the pump.io process is compromised by a remote attacker. However, it *does* introduce problems for anyone who stores data (whether `datadir` or the source code itself) anywhere that `ProtectHome=true` restricts access to, including `/home`, `/root`, and `/run/user`. You will need to move all this data to an accessible location.

Warning: It is perfectly possible to use systemd's unit fragment mechanism to override the pump.io service and set `ProtectHome` to `false`. Administrators are strongly encouraged to avoid this option as there are better solutions and turning it off reduces installation security.

If you store `datadir` in `/home`, the recommended course of action is to move it to `/var/local/pump.io`. Other reasonable choices include `/var/lib/pump.io`, `/srv/pump.io` or `/srv/http/pump.io`. These suggestions are based on `hier(7)` and the Filesystem Hierarchy Standard, but really the place you pick is not terribly important as long as systemd allows access to it. Note that all of both `/var` and `/srv` are left accessible specifically for this purpose. When moving `datadir`, don't forget to adjust your `pump.io.json` to match.

If you have an npm-based install, this is all you need to worry about. However, if you have a source-based install and you store pump.io's source code in a location restricted by systemd, pump.io will be unable to start.

There are two solutions to this:

1. Move the source code somewhere else. `/srv` or `/usr/local/lib` might be good choices for this.
2. Switch to managing pump.io as a global npm install, but installing from your local source code instead of from npm. To do this, make sure you're in the directory with your source code, then run `sudo npm install -g --production .`. You can then manage pump.io as if it was installed from npm, except that the source code will correspond to your local checkout. See npm-install(1) for how this works.

   This will be the recommended from-source install method in the future.

If you have any questions, aren't sure how to pick, or don't understand what's going on, get in touch with the community.

6.4 Routine maintenance

6.4.1 Viewing pump.io logs

pump.io uses Bunyan for its logs.

Bunyan comes with a command-line tool which can format your logs' JSON into something much prettier.

Installing the Bunyan CLI

The Bunyan CLI can be installed with `npm`:

```
$ sudo npm install -g bunyan
```

Viewing your logs

To view your logs, just invoke the Bunyan CLI with the log filename specified in your `pump.io.json`, or pipe from stdin.

For example, to just pretty-print pump.io's output:

```
$ pump 2>&1 | bunyan
```

Note: You need `2>&1` because pump.io prints logs to stderr.

Or, if your logs are stored at `/var/log/pump.io/pump.io.log.json`:

```
$ bunyan /var/log/pump.io/pump.io.log.json
```

Bunyan logfiles can get quite large and `bunyan` can take a while to format large files, so it may be smart to only view the latest logs:

```
$ tail /var/log/pump.io/pump.io.log.json | bunyan
```

Filtering logs

Bunyan logs are structured. You can filter them to only show (for example) certain types of messages using the `-l` flag.

Valid loglevels are `fatal`, `error`, `warn`, `info`, and `debug`. See the Bunyan documentation for details on what these levels mean, and note that pump.io does not use the `trace` loglevel at all.

For example, if we wanted to show only `fatal` and `error` messages:

```
$ bunyan -l error
```

Note: if you're trying to view `debug` messages but aren't seeing any, pump.io is probably not writing `debug` log messages. Try adjusting the `logLevel` config option.

The Bunyan CLI has more options for filtering and output: you can see them by invoking `bunyan --help`.

6.4.2 Using the upstream pump.io systemd unit

pump.io ships a systemd unit file which all administrators are strongly encouraged to use.

This has several benefits:

1. You don't have to write a systemd unit yourself
2. Your node will automatically use production best practices, such as setting `NODE_ENV` to `production` in the environment
3. You automatically get an improved setup when we change the service file to e.g. take advantage of systemd security features

To start using the systemd unit shipped with pump.io, install a symlink in systemd's unit file directory:

Warning: You should double-check that these paths are the correct paths for your system before running this command. In particular, some npm setups may install pump.io to `/usr/local/lib/node_modules` instead of `/usr/lib/node_modules`.

```
$ sudo ln -s /usr/lib/node_modules/pump.io/pump.io@.service /lib/systemd/system/
```

Now that you've got your unit installed, you need to figure out the name of your unit. The pump.io systemd unit is parameterized so that you can specify the systemd service name of your Databank driver. The unit will arrange to have pump.io depend on the driver service being available. (See the "Specifiers" section of systemd.unit(8) for more details on how this works.)

For example, if your Databank driver is MongoDB, your pump.io unit name would be `pump.io@mongodb.service`. This will cause `mongodb.service` to be started before pump.io.

Note: You can override individual service file options using drop-in files.

To actually *enable* the unit, try `systemctl`:

```
$ sudo systemctl enable pump.io@mongodb
```

If that command works, great! You're done. However, if you received `Failed to execute operation: No such file or directory` or something similar, you are likely affected by systemd bug #3010, so you'll need to manually enable the service. This is easy enough with `ln`:

```
$ sudo ln -s /lib/systemd/system/pump.io@.service /etc/systemd/system/multi-user.
↪target.wants/pump.io@mongodb.service
```

pump.io will now automatically start at boot. To start it immediately:

```
$ sudo systemctl daemon-reload
$ sudo systemctl start pump.io@mongodb
```

Congratulations! You're now using the systemd unit file shipped with pump.io.

6.4.3 Using zero-downtime restarts

Since version 5.1, pump.io has the ability to roll over to new codebases or configurations with no impact on uptime.

> **Warning:** You should schedule maintenance windows even when making use of this feature. To preserve stability this feature's error handling is extremely conservative, so if something goes wrong, you will need to restart pump.io the "normal" way.

To make use of this feature, first ensure you meet the requirements:

1. MongoDB as the Databank driver
2. Two or more cluster workers configured (this is the default)
3. pump.io 5.1 or better

When performing a zero-downtime restart, pump.io will abort if it encounters any of the following errors:

- A requirement is not met
- A magic number from the *new* code doesn't match the magic number from the *old* code loaded when the master process started - this number will be incremented for things that would make zero-downtime restarts cause problems; for example:
  - The logic in the master process itself changing
  - Cross-process logic changing, such that a new worker communicating with old workers would cause problems
  - Database changes
- A new worker died directly after being spawned (e.g. from invalid JSON in `pump.io.json`)
- A new worker signaled that it couldn't bind to the appropriate ports

If a zero-downtime restart fails for either of the last two reasons, the master process will refuse subsequent restart requests and will not respawn any more cluster workers. In this case, you should restart your master process as soon as possible.

Note also that if a worker process doesn't shut itself down within 30 seconds, it will be killed, and pump.io will also refuse a restart request if a restart is already in progress.

To prepare for the restart, first start a stream of your logs. For example:

```
$ sudo tail -f /var/log/pump.io/pump.io.log.json | bunyan
```

This step is very important as pump.io will report any errors to the logfile.

To actually trigger a zero-downtime restart, send SIGUSR2 to the pump.io master process. For example:

```
$ sudo killall -USR2 node
```

> **Warning:** Node's default action upon receiving SIGUSR2 is to terminate. pump.io worker processes override this behavior, but other Node.js programs on your system might not. Take care to not signal any processes you don't want to kill.

You should continue to observe your logs until you see a message about the zero-downtime restart being complete.

6.4.4 Migrating hosts

If you ever want to migrate to another server, you need to copy the contents of your Databank backend as well as the `datadir`. Before you perform this procedure, make sure to shut down your pump.io server.

For example, if you're using `databank-mongodb` you could do something like this:

```
$ systemctl stop pump.io@mongodb
$ mongodump
$ rsync -var dump new.server.com:
$ rsync /var/local/pumpio/ new.server.com:

$ ssh new.server.com
$ mongorestore
$ mv pumpio /var/local/pumpio/
```

This example assumes that you're moving to `new.server.com` and that your `datadir` is set to `/var/local/pumpio/`.

See also the Sysadmin FAQ.

CHAPTER 7

Frequently Asked Questions for sysadmins

Some common (and uncommon) issues that can come up while deploying Pump, and some solutions to match.

7.1 Stuff in my brand-new account randomly doesn't work.

7.2 I get Mixed Content Blocker warnings in the browser console.

7.3 Some actions randomly send requests to the wrong port in the devtools network pane.

This is usually caused by a reverse-proxy setup that was botched at some point. The solution is to make sure your URL is exactly the way you want it, including protocol scheme (i.e. HTTP or HTTPS - hopefully the latter!) and port. Then you need to drop your database (how to do this will depend on your Databank driver) and recreate any accounts you may have had on the server.

Note that if you did anything beyond just creating a new account you should choose a new URL that you haven't used yet, if possible.

(The reason that this issue happens is that ActivityStreams objects, which Pump heavily uses in its protocol, contain an id which Pump usually sets to the object's URL. A lot of these objects get created when new accounts are opened, and if this happens while the URL is somehow incorrect, the incorrect URL gets permanently embedded in the ActivityStreams object and doesn't update when the administrator adjusts the site configuration to fix the problem. This causes a myriad of problems which exhibit themselves in strange ways.)

7.4 I set *urlPort*, but my browser is still sending me to the wrong port.

Clear your cache and try again.

7.5 My question isn't covered, or the solution didn't work for me.

Get in touch with the community. They'll be happy to help you out.

CHAPTER 8

For developers

Eventually this section will be much more thorough, but in the meantime you should check out API.md for a description of what the API looks like and what you can do with it.

Note that we are in the process of deprecating this API and moving to ActivityPub as well as OAuth 2.0. If you write things to the API described in `API.md` you shouldn't have too much trouble moving to ActivityPub - ActivityPub is actually derived from the original pump.io API! You can experiment with this code as it lands by running alpha builds.

For now, most of the documentation is at the old wiki.

Check out the community information.